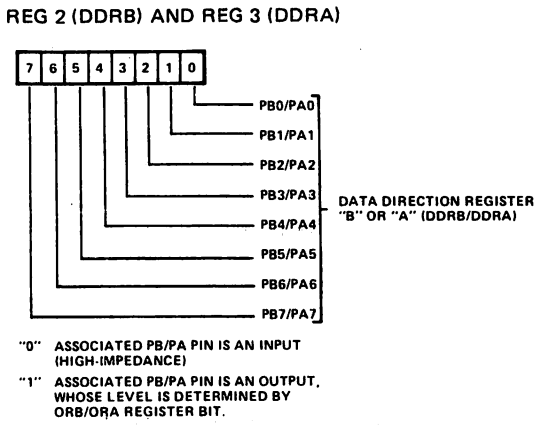
Figure H-11: Data Direction Registers (DDRB, DDRA)

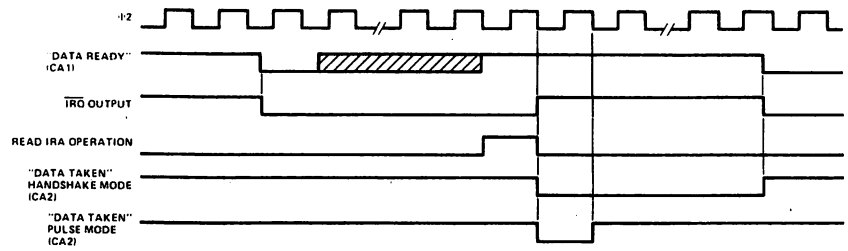


READ HANDSHAKE

Positive control of data transfers from peripheral devices into the system processor can be accomplished very effectively using Read Handshaking. In this case, the peripheral device must generate the equivalent of a "Data Ready" signal to the processor signifying that valid data is present on the peripheral port. This signal normally interrupts the processor, which then reads the data, causing generation of a "Data Taken" signal. The peripheral device responds by making new data available. This process continues until the data transfer is complete.

In the SY6522, automatic "Read" Handshaking is possible on the Peripheral A port only. The CA1 interrupt input pin accepts the "Data Ready" signal and CA2 generates the "Data Taken" signal. The "Data Ready" signal will set an internal flag which may interrupt the processor or which may be polled under program control. The "Data Taken" signal can either be a pulse or a level which is set low by the system processor and is cleared by the "Data Ready" signal. These options are shown in Figure H-12, which illustrates the normal Read Handshaking sequence.

Figure H-12: Read Handshake Timing (Port A, Only)

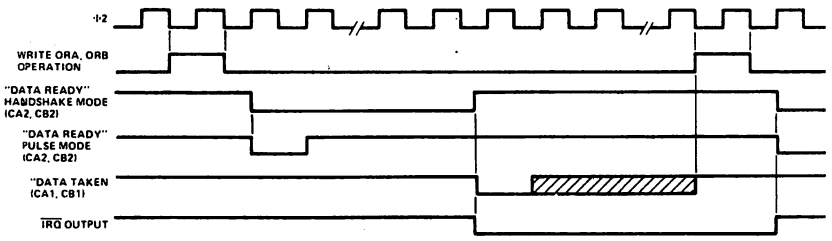


WRITE HANDSHAKE

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described for Read Handshaking. However, for Write Handshaking, the SY6522 generates the "Data Ready" signal and the peripheral device must respond with the "Data Taken" signal. This can be accomplished on both the PA port and the PB port on the SY6522. CA2 or CB2 act as a "Data Ready" output in either the handshake mode or pulse mode and CA1 or CB1 accept the "Data Taken" signal from the peripheral device, setting the Interrupt Flag and cleaning the "Data Ready" output. This sequence is shown in Figure H-13.

Selection of operating modes for CA1, CA2, CB1, and CB2 is accomplished by the Peripheral Control Register (Figure H-14).

Figure H-13: Write Handshake Timing

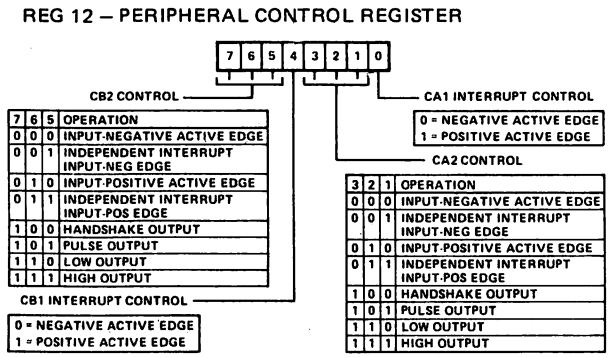


TIMER OPERATION

Interval Timer T1 consists of two 8-bit latches and a 16-bit counter. The latches are used to store data which is to be loaded into the counter. After loading, the counter decrements at $\phi 2$ clock rate. Upon reaching zero, an Interrupt Flag will be set, and IRQ will go low if the interrupt is enabled. The timer will then disable any further interrupts, or will automatically transfer the contents of the latches into the counter and will continue to decrement. In addition, the timer may be programmed to invert the output signal on a peripheral pin each time it "times-out". Each of these modes is discussed separately below.

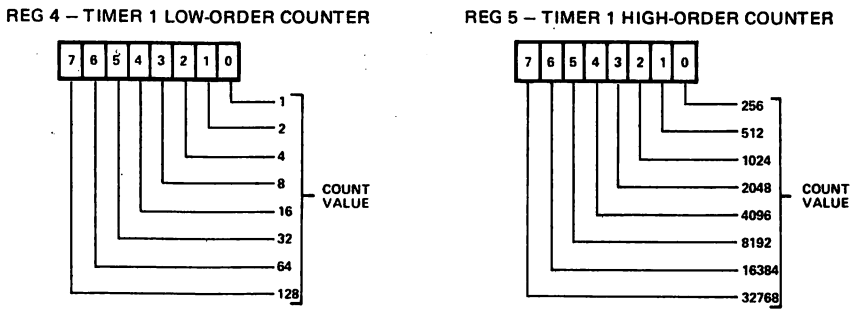
The T1 counter is depicted in Figure H-15 and the latches in Figure H-16.

Figure H-14: CA1, CA2, CB1, CB2 Control



Two bits are provided in the Auxiliary Control Register (bits 6 and 7) to allow selection of the T1 operating modes. The four possible modes are depicted in Figure H-17.

Figure H-15: T1 Counter Registers



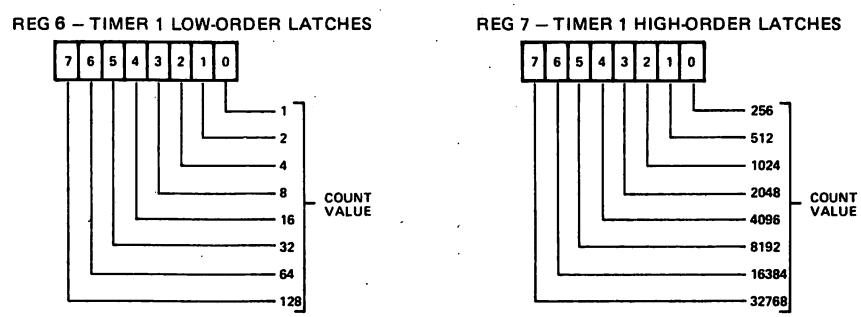
WRITE – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. LATCH CONTENTS ARE TRANSFERRED INTO LOW-ORDER COUNTER AT THE TIME THE HIGH-ORDER COUNTER IS LOADED (REG 5).

READ – 8 BITS FROM T1 LOW-ORDER COUNTER TRANSFERRED TO MPU. IN ADDITION, T1 INTERRUPT FLAG IS RESET (BIT 6 IN INTERRUPT FLAG REGISTER).

WRITE – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. ALSO, AT THIS TIME BOTH HIGH AND LOW-ORDER LATCHES TRANSFERRED INTO T1 COUNTER. T1 INTERRUPT FLAG ALSO IS RESET.

READ – 8 BITS FROM T1 HIGH-ORDER COUNTER TRANSFERRED TO MPU.

Figure H-16: T1 Latch Registers



WRITE – 8 BITS LOADED INTO T1 LOW-ORDER LATCHES. THIS OPERATION IS NO DIFFERENT THAN A WRITE INTO REG 4.

READ – 8 BITS FROM T1 LOW-ORDER LATCHES TRANSFERRED TO MPU. UNLIKE REG 4 OPERATION, THIS DOES NOT CAUSE RESET OF T1 INTERRUPT FLAG.

WRITE – 8 BITS LOADED INTO T1 HIGH-ORDER LATCHES. UNLIKE REG 4 OPERATION NO LATCH-TO-COUNTER TRANSFERS TAKE PLACE.

READ – 8 BITS FROM T1 HIGH-ORDER LATCHES TRANSFERRED TO MPU.

Figure H-17: Auxiliary Control Register

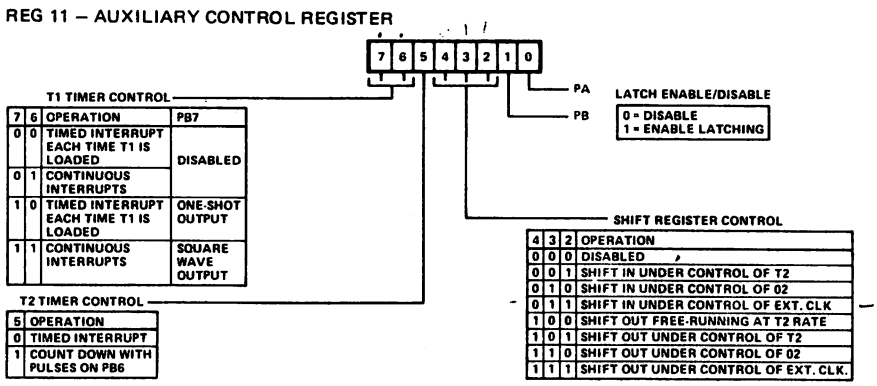
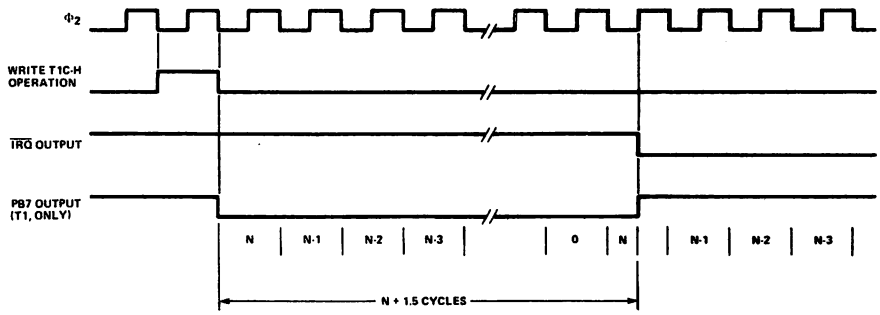


Figure H-18: Timer 1 and Timer 2 One-Shot Mode Timing



TIMER 1 ONE-SHOT MODE

The interval timer one-shot mode allows generation of a single interrupt for each timer load operation. As with any interval timer, the delay between the "write T1C-H" operation and generation of the processor interrupt is a direct function of the data loaded into the timing counter. In addition to generating a single interrupt, Timer 1 can be programmed to produce a single negative pulse on the PB7 peripheral pin. With the output enabled (ACR7 = 1) a "write T1C-H" operation will cause PB7 to go low. PB7 will return high when Timer 1 times out. The result is a single programmable width pulse.

In the one-shot mode, writing into the high order latch has no effect on the operation of Timer 1. However, it will be necessary to assure that the low order latch contains the proper data before initiating the count-down with a "write T1C-H" operation. When the processor writes into the high order counter, the T1 Interrupt Flag will be cleared, the contents of the low order latch will be transferred into the low order counter, and the timer will begin to decrement at system clock rate. If the PB7 output is enabled, this signal will go low on the phase two following the write operation. When the counter reaches zero, the T1 Interrupt Flag will be set, the IRQ pin will go low

(interrupt enabled), and the signal on PB7 will go high. At this time the counter will continue to decrement at system clock rate. This allows the system processor to read the contents of the counter to determine the time since interrupt. However, the T1 Interrupt Flag cannot be set again unless it has been cleared as described in this specification.

Timing for the SY6522 interval timer one-shot modes is shown in Figure H-18.

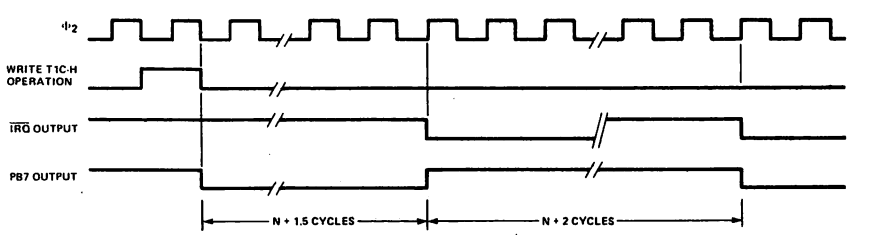
TIMER 1 FREE-RUN MODE

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and the ability to produce a square wave on PB7 whose frequency is not affected by variations in the processor interrupt response time. This is accomplished in the "free-running" mode.

In the free-running mode, the Interrupt Flag is set and the signal on PB7 is inverted each time the counter reaches zero. However, instead of continuing to decrement from zero after a time-out, the timer automatically transfers the contents of the latch into the counter (16 bits) and continues to decrement from there. The Interrupt Flag can be cleared by writing T1C-H, by reading T1C-L, or by writing directly into the flag as described later. However, it is not necessary to rewrite the timer to enable setting the Interrupt Flag on the next time-out.

All interval timers in the SY6522 are "re-triggerable". Rewriting the counter will always re-initialize the time-out period. In fact, the time-out can be prevented completely if the processor continues to rewrite the timer before it reaches zero. Timer 1 will operate in this manner if the processor writes into the high order counter (T1C-H). However, by loading the latches only, the processor can access the timer during each down-counting operation without affecting the time-out in process. Instead, the data loaded into the latches will determine the length of the next time-out period. This capability is particularly valuable in the free-running mode with the output enabled. In this mode, the signal on PB7 is inverted and the interrupt flag is set with each time-out. By responding to the interrupts with new data for the latches, the processor can determine the period of the next half cycle during each half cycle of the output signal on PB7. In this manner, very complex waveforms can be generated. Timing for the free-running mode is shown in Figure H-19.

Figure H-19: Timer 1 Free-Run Mode Timing



Note: A precaution to take in the use of PB7 as the timer output concerns the Data Direction Register contents for PB7. Both DDRB bit 7 and ACR bit 7 must be 1 for PB7 to function as the timer output. If one is 1 and the other is 0, then PB7 functions as a normal output pin, controlled by ORB bit 7.

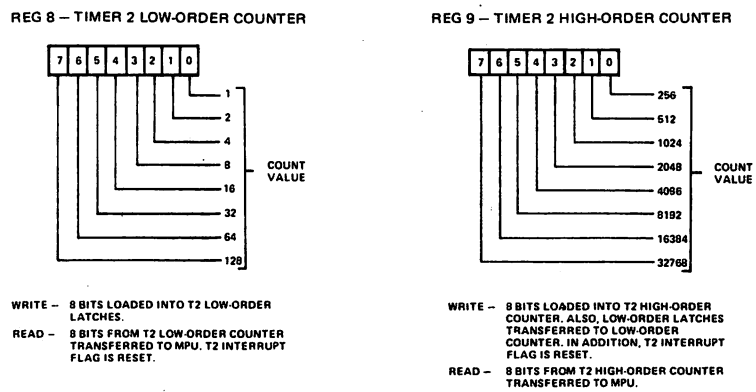
TIMER 2 OPERATION

Timer 2 operates as an interval timer (in the "one-slot" mode only), or as a counter for counting negative pulses on the PB6 peripheral pin. A single control bit is provided in the Auxiliary Control Register to select between these two modes. This timer is comprised of a "write-only" low-order latch (T2L-L), a "read-only" low-order counter and a read/write high order counter. The counter registers act as a 16-bit counter which decrements at $\phi 2$ rate. Figure H-20 illustrates the T2 Counter Registers.

TIMER 2 ONE-SHOT MODE

As an interval timer, T2 operates in the "one-shot" mode similar to Timer 1. In this mode, T2 provides a single interrupt for each "write T2C-H" operation. After timing out, the counter will continue to decrement. However, setting of the Interrupt Flag will be disabled after initial time-out so that it will not be set by the counter continuing to decrement through zero. The processor must rewrite T2C-H to enable setting of the Interrupt Flag. The Interrupt Flag is cleared by reading T2C-L or by writing T2C-H. Timing for this operation is shown in Figure H-18.

Figure H-20: T2 Counter Registers



TIMER 2 PULSE COUNTING MODE

In the pulse counting mode, T2 serves primarily to count a predetermined number of negative-going pulses on PB6. This is accomplished by first loading a number into T2. Writing into T2C-H clears the Interrupt Flag and allows the counter to decrement each time a pulse is applied to PB6. The Interrupr Flag will be set when T2 reaches zero. At this time the counter will continue to decrement with each pulse on PB6. However, it is necessary to rewrite T2C-H to allow the Interrupt Flag to set on subsequent down-counting operations. Timing for this mode is shown in Figure H-21. The pulse must be low on the leading edge of $\phi 2$.

SHIFT REGISTER OPERATION

The Shift Register (SR) performs serial data transfers into and out of the CB2 pin under control of an internal modulo-8 counter. Shift pulses can be applied to the CB1 pin from an external source or, with the proper mode selection, shift pulses generated internally will appear on the CB1 pin for controlling external devices.

The control bits which select the various shift register operating modes are located in the Auxiliary Control Register. Figure H-22 illustrates the configuration of the SR data bits and the SR control bits of the ACR.

Figures H-23 and H-24 illustrate the operation of the various shift register modes.

INTERRUPT OPERATION

Controlling interrupts within the SY6522 involves three principle operations. These are flagging the interrupts, enabling interrupts and signaling to the processor that an active interrupt exists within the chip. Interrupt flags are set by interrupting conditions which exist within the chip or on inputs to the chip. These flags normally remain set until the interrupt has been serviced. To determine the source of an interrupt, the microprocessor must examine these flags in order from highest to lowest priority. This is accomplished by reading the flag register into the processor accumulator, shifting this register either right or left and then using conditional branch instructions to detect an active interrupt.

Associated with each Interrupt Flag is an interrupt enable bit. This can be set or cleared by the processor to enable interrupting the processor from the corresponding Interrupt Flag. If an interrupt flag is set to a logic 1 by an interrupting condition, and the corresponding interrupt enable bit is set to a 1, the Interrupt Request Output (IRQ) will go low. IRQ is an "open-collector" output which can be "wire-or'ed" with other devices in the system to interrupt the processor.

In the SY6522, all the Interrupt Flags are contained in one register. In addition, bit 7 of this register will be read as a logic 1 when an interrupt exists within the chip. This allows very convenient polling of several devices within a system to locate the source of an interrupt.

Figure H-21: Timer 2 Pulse Counting Mode

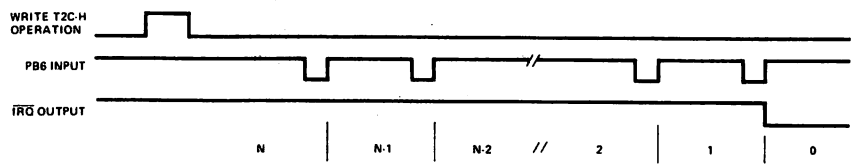
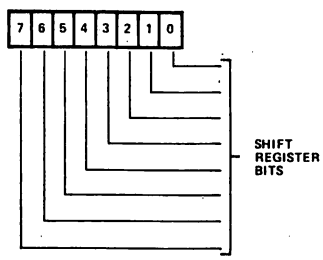


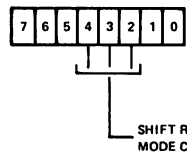
Figure H-22: SR and ACR Control Bits

REG 10 – SHIFT REGISTER



NOTES:
1. WHEN SHIFTING OUT, BIT 7 IS THE FIRST BIT OUT AND SIMULTANEOUSLY IS ROTATED BACK INTO BIT 0.
2. WHEN SHIFTING IN, BITS INITIALLY ENTER BIT 0 AND ARE SHIFTED TOWARDS BIT 7.

REG 11 – AUXILIARY CONTROL REGISTER



4	3	2	OPERATION
0	0	0	DISABLED
0	0	1	SHIFT IN UNDER CONTROL OF T2
0	1	0	SHIFT IN UNDER CONTROL OF Φ_2
0	1	1	SHIFT IN UNDER CONTROL OF EXT CLK
1	0	0	SHIFT OUT FREE-RUNNING AT T2 RATE
1	0	1	SHIFT OUT UNDER CONTROL OF T2
1	1	0	SHIFT OUT UNDER CONTROL OF Φ_2
1	1	1	SHIFT OUT UNDER CONTROL OF EXT CLK

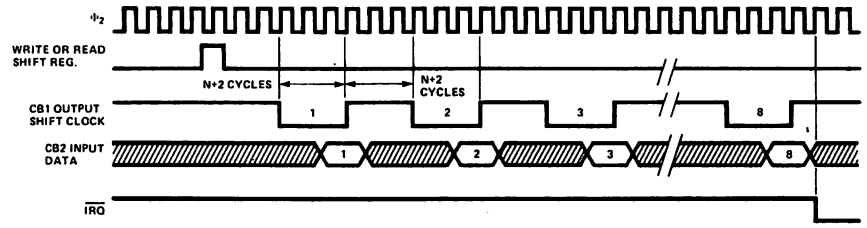
SR Disabled (000)

The 000 mode is used to disable the Shift Register. In this mode the microprocessor can write or read the SR, but the shifting operation is disabled and operation of CB1 and CB2 is controlled by the appropriate bits in the Peripheral Control Register (PCR). In this mode the SR Interrupt Flag is disabled (held to a logic 0).

Shift in Under Control of T2 (001)

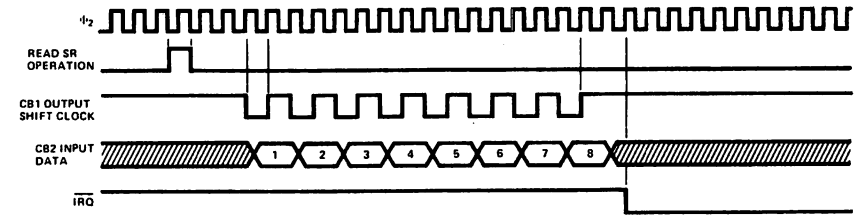
In the 001 mode the shifting rate is controlled by the low order 8 bits of T2. Shift pulses are generated on the CB1 pin to control shifting in external devices. The time between transitions of this output clock is a function of the system clock period and the contents of the low order T2 latch (N).

The shifting operation is triggered by writing or reading the shift register. Data is shifted first into the low order bit of SR and is then shifted into the next higher order bit of the shift register on the negative-going edge of each clock pulse. The input data should change before the positive-going edge of the CB1 clock pulse. This data is shifted into the shift register during the ϕ_2 clock cycle following the positive-going edge of the CB1 clock pulse. After 8 CB1 clock pulses, the shift register Interrupt Flag will be set and IRQ will go low.



Shift in Under Control of ϕ_2 (010)

In mode 010 the shift rate is a direct function of the system clock frequency. CB1 becomes an output which generates shift pulses for controlling external devices. Timer 2 operates as an independent interval timer and has no effect on SR. The shifting operation is triggered by reading or writing the Shift Register. Data is shifted first bit 0 and is then shifted into the next higher order bit of the shift register on the trailing edge of each ϕ_2 clock pulse. After 8 clock pulses, the shift register Interrupt Flag will be set, and the output clock pulses on CB1 will stop.

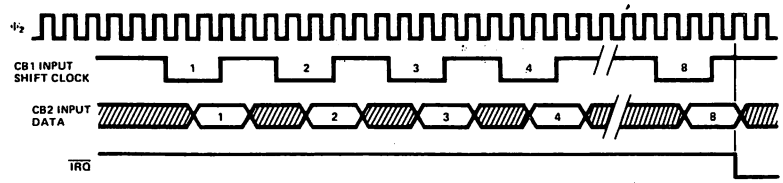


Shift in Under Control of External CB1 Clock (011)

In mode 011 CB1 becomes an input. This allows an external device to load the shift register at its own pace. The shift register counter will interrupt the processor each time 8 bits have been shifted in. However, the shift register counter does not stop the shifting operation; it acts simply as a pulse counter. Reading or writing the Shift Register resets the Interrupt flag and initializes the SR counter to count another 8 pulses.

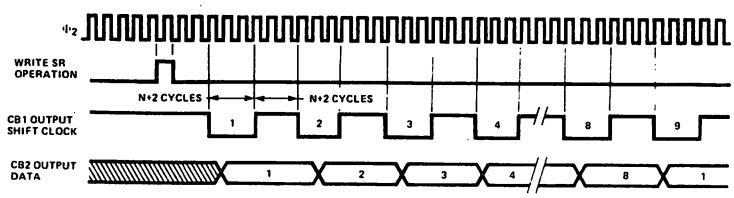
Note that the data is shifted during the first system clock cycle following the positive-going edge of the CB1 shift pulse. For this reason, data must be held stable during the first full cycle following CB1 going high.

Figure H-23: Shift Register Input Modes



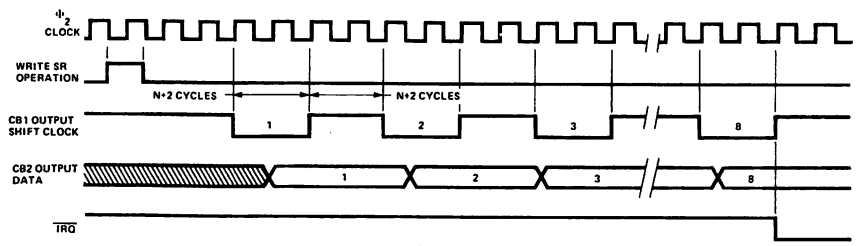
Shift Out Free-Running at T2 Rate (100)

Mode 100 is very similar to mode 101 in which the shifting rate is set by T2. However, in mode 100 the SR counter does not stop the shifting operation. Since the Shift Register bit 7 (SR7) is recirculated back into bit 0, the 8 bits loaded into the shift register will be clocked onto CB2 repetitively. In this mode the shift register counter is disabled.



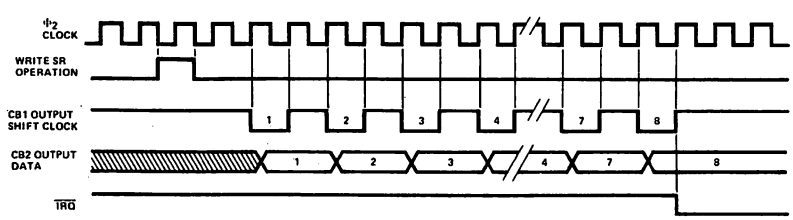
Shift Out Under Control of T2 (101)

In mode 101 the shift rate is controlled by T2 (as in the previous mode). However, with each read or write of the shift register the SR Counter is reset and 8 bits are shifted onto CB2. At the same time, 8 shift pulses are generated on CB1 to control shifting in external devices. After the 8 shift pulses, the shifting is disabled, the SR Interrupt Flag is set and CB2 remains at the last data level.



Shift Out Under Control of ϕ_2 (110)

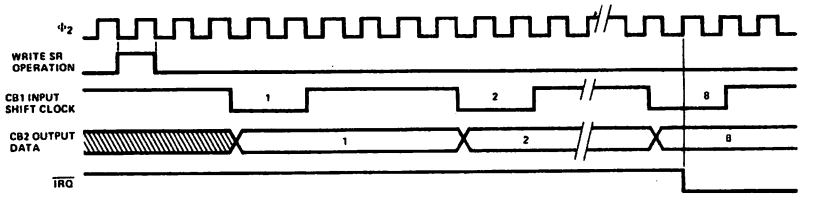
In mode 110, the shift rate is controlled by the ϕ_2 system clock.



Shift Out Under Control of External CB1 Clock (111)

In mode 111 shifting is controlled by pulses applied to the CB1 pin by an external device. The SR counter sets the SR Interrupt flag each time it counts 8 pulses but it does not disable the shifting function. Each time the microprocessor writes or reads the shift register, the SR Interrupt flag is reset and the SR counter is initialized to begin counting the next 8 shift pulses on pin CB1. After 8 shift pulses, the Interrupt flag is set. The microprocessor can then load the shift register with the next byte of data.

Figure H-24: Shift Register Output Modes



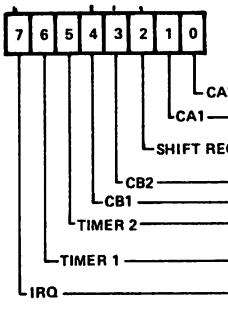
The Interrupt Flag Register (IFR) and Interrupt Enable Register (IER) are depicted in Figures H-25 and H-26, respectively.

The IFT may be read directly by the processor. In addition, individual flag bits may be cleared by writing a "1" into the appropriate bit of the IFR. When the proper chip select and register signals are applied to the chip, the contents of this register are placed on the data bus. Bit 7 indicates the status of the IRQ output. This bit corresponds to the logic function: IRQ = IFR6xIER6 + IFR5x IER5 + IFR4xIER4 + IFR3xIER3 + IFR2xIER2 + IFR1xIER1 + IFR0xIER0. Note: X = logic AND, + = Logic OR.

The IFR bit 7 is not a flag. Therefore, this bit is not directly cleared by writing a logic 1 into it. It can only be cleared by clearing all the flags in the register or by disabling all the active interrupts as discussed in the next section.

Figure H-25: Interrupt Flag Register (IFR)

REG 13 – INTERRUPT FLAG REGISTER



	SET BY	CLEARED BY
CA2	CA2 ACTIVE EDGE	READ OR WRITE REG 1 (ORA)*
CA1	CA1 ACTIVE EDGE	READ OR WRITE REG 1 (ORA)
SHIFT REG	COMPLETE 8 SHIFTS	READ OR WRITE SHIFT REG
CB2	CB2 ACTIVE EDGE	READ OR WRITE ORB*
CB1	CB1 ACTIVE EDGE	READ OR WRITE ORB
TIMER 2	TIME-OUT OF T2	READ T2 LOW OR WRITE T2 HIGH
TIMER 1	TIME-OUT OF T1	READ T1 LOW OR WRITE T1 HIGH
IRQ	ANY ENABLED INTERRUPT	CLEAR ALL INTERRUPTS

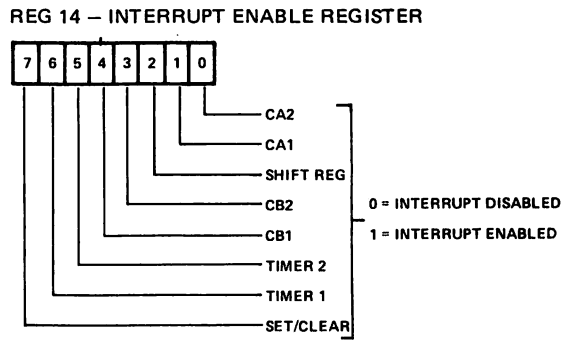
* IF THE CA2/CB2 CONTROL IN THE PCR IS SELECTED AS "INDEPENDENT" INTERRUPT INPUT, THEN READING OR WRITING THE OUTPUT REGISTER ORA/ORB WILL NOT CLEAR THE FLAG BIT. INSTEAD, THE BIT MUST BE CLEARED BY WRITING INTO THE IFR, AS DESCRIBED PREVIOUSLY.

For each Interrupt Flag in IFR, there is a corresponding bit in the Interrupt Enable Register. The system processor can be set or clear selected bits in this register to facilitate controlling individual interrupts without affecting others. This is accomplished by writing to address 1110 (IER address). If bit 7 of the data placed on the system data bus during this write operation is a 0, each 1 in bits 6 through 0 clears the corresponding bit in the Interrupt Enable Register. For each zero in bits 6 through 0, the corresponding bit is unaffected.

Setting selected bits in the Interrupt Enable Register is accomplished by writing to the same address with bit 7 in the data word set to a logic 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each zero, the corresponding bit will be unaffected. This individual control of the setting and clearing operations allows very convenient control of the interrupts during system operation.

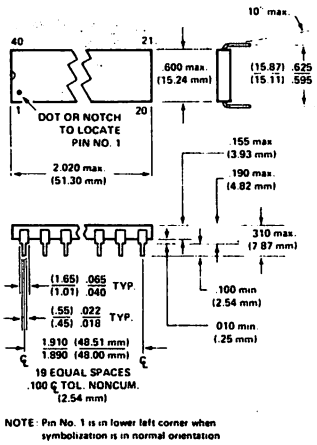
In addition to setting and clearing IER bits, the processor can read the contents of this register by placing the proper address on the register select and chip select inputs with the R/W line high. Bit 7 will be read as a logic 0.

Figure H-26: Interrupt Enable Register (IER)



NOTES:
1. IF BIT 7 IS A "0", THEN EACH "1" IN BITS 0 - 6 DISABLES THE CORRESPONDING INTERRUPT.
2. IF BIT 7 IS A "1", THEN EACH "1" IN BITS 0 - 6 ENABLES THE CORRESPONDING INTERRUPT.
3. IF A READ OF THIS REGISTER IS DONE, BIT 7 WILL BE "0" AND ALL OTHER BITS WILL REFLECT THEIR ENABLE/DISABLE STATE.

PACKAGE OUTLINE

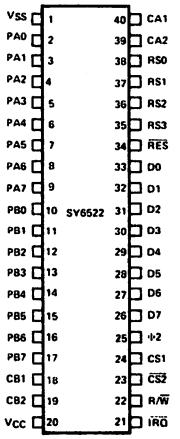


NOTE: Pin No. 1 is in lower left corner when symbolization is in normal orientation

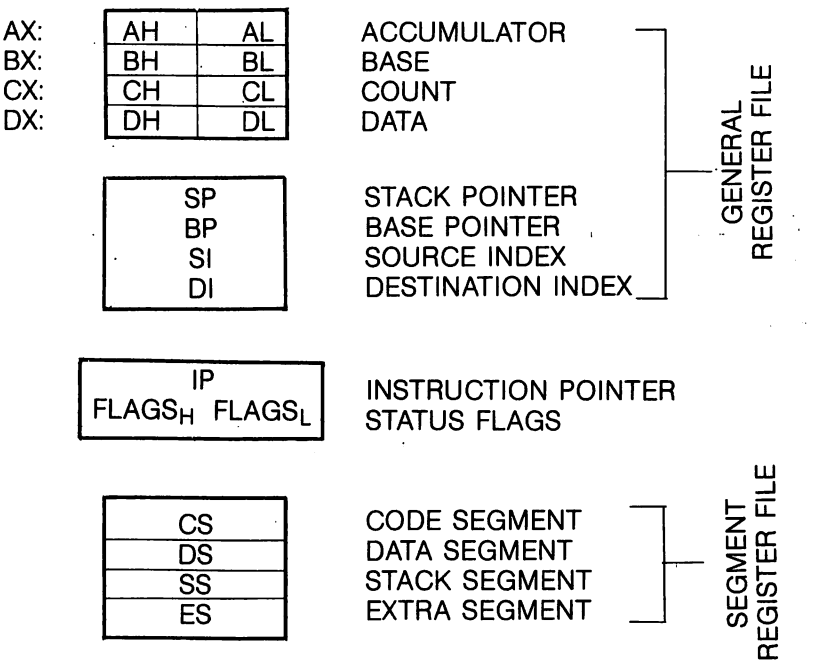
ORDERING INFORMATION

Order Number	Package Type	Frequency Option
SYP 6522	Plastic	1 MHz
SYP 6522A	Plastic	2 MHz
SYC 6522	Ceramic	1 MHz
SYC 6522A	Ceramic	2 MHz

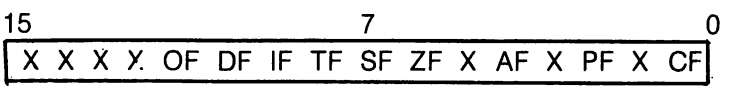
PIN CONFIGURATION



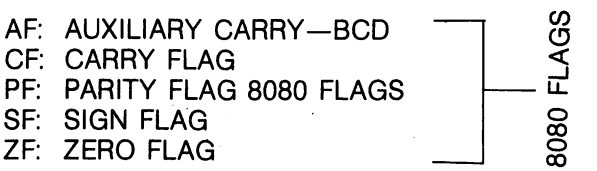
I.1 8086 REGISTER MODEL

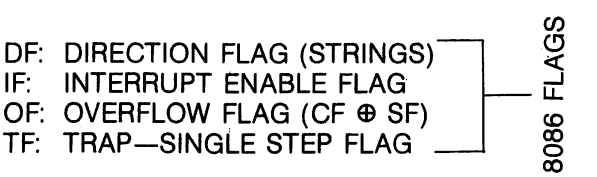


Instructions which reference the flag register file as a 16-bit object use the symbol FLAGS to represent the file:



X = Don't care





I.2 OPERAND SUMMARY

"REG" FIELD BIT ASSIGNMENTS

16-BIT(W=1)	8-BIT(W=0)	SEGMENT
000 AX	000 AL	00 ES
001 CX	001 CL	01 CS
010 DX	010 DL	10 SS
011 BX	011 BL	11 DS
100 SP	100 AH	
101 BP	101 CH	
110 SI	110 DH	
111 DI	111 BH	

I.3 SECOND INSTRUCTION BYTE SUMMARY

mod xxx r/m

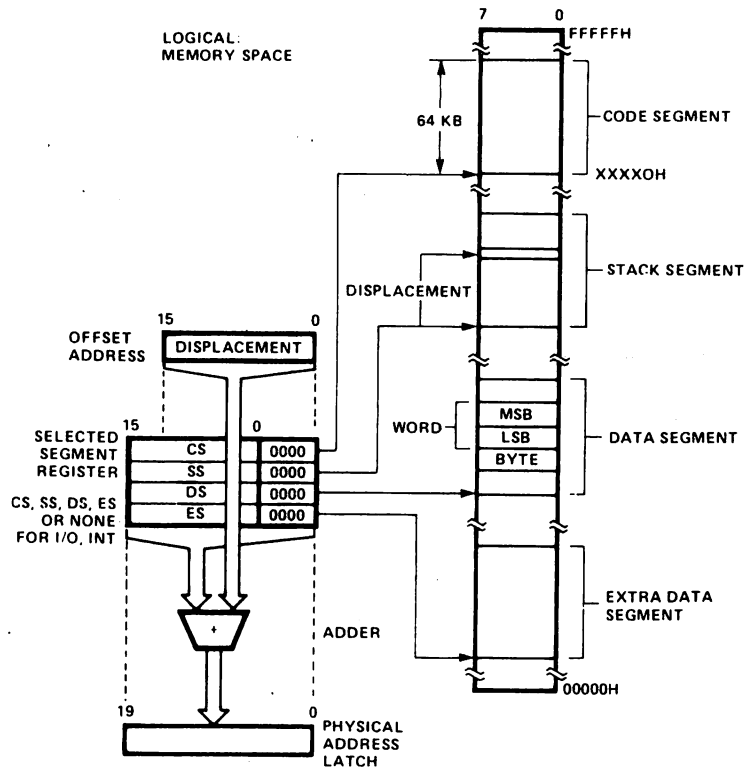
MOD	DISPLACEMENT
00	DISP=0*; disp-low and disp-high are absent
01	DISP=disp-low sign-extended to 16-bits, disp-high is absent
10	DISP=disp-high:disp-low
11	r/m is treated as a "reg" field

R/M	OPERAND ADDRESS	DEFAULT SEGMENT
000	(BX) + (SI) + DISP	DS
001	(BX) + (DI) + DISP	DS
010	(BP) + (SI) + DISP	SS
011	(BP) + (DI) + DISP	SS
100	(SI) + DISP	DS
101	(DI) + DISP	DS
110	(BP) + DISP*	SS
111	(BX) + DISP	DS

DISP follows 2nd byte of instruction (before data if required).
*except if mod=00 and r/m=110; then EA=disp-high: disp-low.

OPERAND ADDRESS (EA) TIMING (CLOCKS):
Add 4 clocks for word operands at ODD ADDRESSES.
Immed offset=6
Base (BX, BP, SI, DI)=5
Base + DISP=9
Base + index (BP + DI, BX + SI)=7
Base + index (BP + SI, BX + DI)=8
Base + index (BP + DI, BX + SI) + DISP=11
Base + index (BP + SI, BX + DI) + DISP=12

I.4 MEMORY SEGMENTATION MODEL



SEGMENT OVERRIDE PREFIX

0 0 1	REG	1 1 0

Timing: 2 clocks

USE OF SEGMENT OVERRIDE

OPERAND REGISTER	DEFAULT	WITH OVERRIDE PREFIX
IP (code address)	CS	Never
SP (stack address)	SS	Never
BP (stack address or stack marker)	SS	BP + DS, or ES, or CS
SI or DI (not incl. strings)	DS	ES, SS, or CS
SI (implicit source addr. for strings)	DS	ES, SS, or CS
DI (implicit dest. addr. for strings)	ES	Never

I.5 INSTRUCTION SET DATA

Section I.5.2 presents instuction set data, grouped by function. Section I.9 provides an alphabetic index to the data.

I.5.1 KEY TO FLAG EFFECTS

The following key refers to the flag sections in the instruction set data in Section I.5.2.

FLAG EFFECT KEY

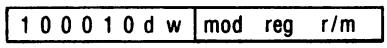
IDENTIFIER	EXPLANATION
(blank)	Not altered
0	Cleared to 0
1	Set to 1
X	Set or cleared according to result
U	Undefined—contains no reliable value
R	Restored from previously-saved value

I.5.2 DATA TRANSFER

MOV=Move

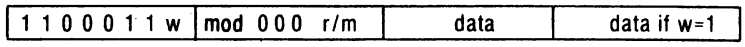
Flags: O D I T S Z A P C

Register/memory to/from register



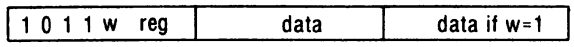
Timing (clocks): register to register 2
memory to register 8+EA
register to memory 9+EA

Immediate to register/memory



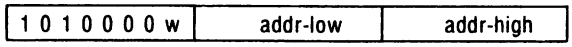
Timing: 10+EA clocks

Immediate to register



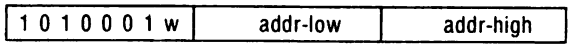
Timing: 4 clocks

Memory to accumulator



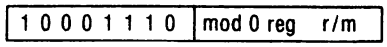
Timing: 10 clocks

Accumulator to memory



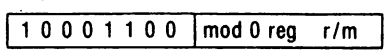
Timing: 10 clocks

Register/memory to segment register



Timing (clocks): register to register 2
memory to register 8+EA

Segment register to register/memory



Timing (clocks): register to register 2
register to memory 9+EA

PUSH=Push

Flags: O D I T S Z A P C

Register/memory

1 1 1 1 1 1 1 1	mod 1 1 0 r/m

Timing (clocks): register 10
memory 16+EA

0 1 0 1 0 reg

Timing: 10 clocks

Segment register

0 0 0 reg 1 1 0

Timing: 10 clocks

POP=Pop

Flags: O D I T S Z A P C

Register/memory

1 0 0 0 1 1 1 1	mod 0 0 0 r/m

Timing (clocks): register 8
memory 17+EA

Register

0 1 0 1 1 reg

Timing: 8 clocks

Segment register

0 0 0 reg 1 1 1

Timing: 8 clocks

XCHG=Exchange

Flags: O D I T S Z A P C

Register/memory with register

1 0 0 0 0 1 1 w	mod reg r/m

Timing (clocks): register with register 4
memory with register 17+EA

Register with accumulator

1 0 0 1 0 reg

Timing: 3 clocks

IN=Input to AL/AX from

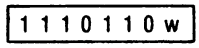
Flags: O D I T S Z A P C

Fixed Port

1 1 1 0 0 1 0 w	port

Timing: 10 clocks

Variable port (DX)

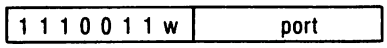


Timing: 8 clocks

OUT=Output from AL/AX to

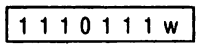
Flags: O D I T S Z A P C

Fixed Port



Timing: 10 clocks

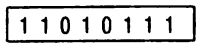
Variable port (DX)



Timing: 8 clocks

XLAT=Translate Byte to AL

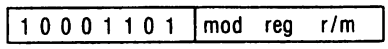
Flags: O D I T S Z A P C



Timing: 11 clocks

LEA=Load EA to Register

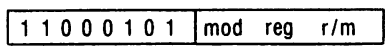
Flags: O D I T S Z A P C



Timing: 2+EA clocks

LDS=Load Pointer to DS

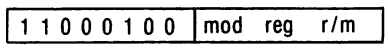
Flags: O D I T S Z A P C



Timing: 16+EA clocks

LES=Load Pointer to ES

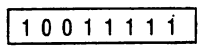
Flags: O D I T S Z A P C



Timing: 16+EA clocks

LAHF=Load AH with Flags

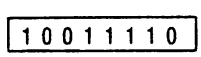
Flags: O D I T S Z A P C



Timing: 4 clocks

SAHF=Store AH into Flags

Flags: O D I T S Z A P C
R R R R R



Timing: 4 clocks

PUSHF=Push Flags

Flags: O D I T S Z A P C

1 0 0 1 1 1 0 0

Timing: 10 clocks

POPF=Pop Flags

Flags: O D I T S Z A P C
R R R R R R R R R

1 0 0 1 1 1 0 1

Timing: 8 clocks

I.5.3 ARITHMETIC

ADD=Add

Flags: O D I T S Z A P C
X X X X X X

Reg./memory with register to either

0 0 0 0 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s:w=01

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 0 0 0 1 0 w	data	data if w=1

Timing: 4 clocks

ADC=Add with Carry

Flags: O D I T S Z A P C
X X X X X X

Reg./memory with register to either

0 0 0 1 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s:w=01

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 0 1 0 1 0 w	data	data if w=1

Timing: 4 clocks

INC=Increment

Flags: O D I T S Z A P C
X X X X X X

Register/memory

1 1 1 1 1 1 1 w	mod 0 0 0 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 0 reg

Timing: 2 clocks

AAA=ASCII Adjust for Add

Flags: O D I T S Z A P C
U U U X U X

0 0 1 1 0 1 1 1

Timing: 4 clocks

DAA=Decimal Adjust for Add

Flags: O D I T S Z A P C
X X X X X X

0 0 1 0 0 1 1 1

Timing: 4 clocks

SUB=Subtract

Flags: O D I T S Z A P C
X X X X X X

0 0 1 0 1 0 d w	mod reg r/m

Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EZ

Immediate from register/memory

1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s:w=01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

0 0 1 0 1 1 0 w	data	data if w=1

Timing: 4 clocks

SBB=Subtract with Borrow

Flags: O D I T S Z A P C
X X X X X

0 0 0 1 1 0 d w	mod reg r/m

Timing (clocks): register from register 3
memory from register 9+EA
register from memory 16+EA

Immediate from register/memory

1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s:w=01

Timing (clocks): immediate from register 4
immediate from memory 17+EA

Immediate from accumulator

0 0 0 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

DEC=Decrement

Flags: O D I T S Z A P C
X X X X X

Register/memory

1 1 1 1 1 1 1 w	mod 0 0 1 r/m

Timing (clocks): register 2
memory 15+EA

Register

0 1 0 0 1 reg

Timing: 2 clocks

NEG=Change Sign

Flags: O D I T S Z A P C
X X X X X 1*

*0 if destination=0

1 1 1 1 0 1 1 w	mod 0 1 1 r/m

Timing (clocks): register 3
memory 16+EA

CMP=Compare

Flags: O D I T S Z A P C
X X X X X X

Register/memory and register

0 0 1 1 1 0 d w	mod reg r/m

Timing (clocks): register with register 3
memory with register 9+EA
register with memory 9+EA

Immediate with register/memory

1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s:w=01

Timing (clocks): immediate with register 4
immediate with memory 17+EA

Immediate with accumulator

0 0 1 1 1 1 0 w	data	data if w=1

Timing: 4 clocks

AAS=ASCII Adjust for Subtract

Flags:

O	D	I	T	S	Z	A	P	C
U				U	U	X	U	X

0 0 1 1 1 1 1 1

Timing: 4 clocks

DAS=Decimal Adjust for Subtract

Flags:

O	D	I	T	S	Z	A	P	C
U				X	X	X	X	X

0 0 1 0 1 1 1 1

Timing: 4 clocks

MUL=Multiply (Unsigned)

Flags:

O	D	I	T	S	Z	A	P	C
X				U	U	U	U	X

1 1 1 1 0 1 1 w	mod 1 0 0 r/m

Timing (clocks): 8-bit 71+EA
16-bit 124+EA

IMUL=Integer Multiply (Signed)

Flags:

O	D	I	T	S	Z	A	P	C
X				U	U	U	U	X

1 1 1 1 0 1 1 w	mod 1 0 1 r/m

Timing (clocks): 8-bit 90+EA
16-bit 144+EA

AAM=ASCII Adjust for Multiply

Flags:

O	D	I	T	S	Z	A	P	C
U				X	X	U	X	U

1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0

Timing: 83 clocks

DIV=Divide (Unsigned)

Flags:

O	D	I	T	S	Z	A	P	C
U				U	U	U	U	U

1 1 1 1 0 1 1 w	mod 1 1 0 r/m

Timing (clocks): 8-bit 90+EA
16-bit 155+EA

IDIV=Integer Divide (Signed)

Flags: O D I T S Z A P C
U U U U U U

1 1 1 1 0 1 1 w	mod 1 1 1 r/m

Timing (clocks): 8-bit 112+EA
16-bit 177+EA

AAD=ASCII Adjust for Divide

Flags: O D I T S Z A P C
U X X U X U

1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0

Timing: 60 clocks

CBW=Convert Byte to Word

Flags: O D I T S Z A P C

1 0 0 1 1 0 0 0

Timing: 2 clocks

CWD=Convert Word to Double Word

Flags: O D I T S Z A P C

1 0 0 1 1 0 0 1

Timing: 5 clocks

I.5.4 LOGIC

NOT=Invert

Flags: O D I T S Z A P C

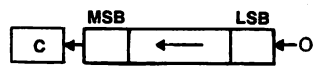
1 1 1 1 0 1 1 w	mod 0 1 0 r/m

Timing (clocks): register 3
memory 16+EA

SHL/SAL=Shift Logical/Arithmetic Left

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 1 0 0 r/m

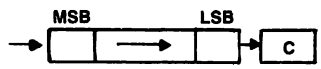


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

SHR=Shift Logical Right

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 1 0 1 r/m

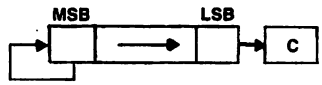


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

SAR=Shift Arithmetic Right

Flags: O D I T S Z A P C
X X X U X X

1 1 0 1 0 0 v w	mod 1 1 1 r/m

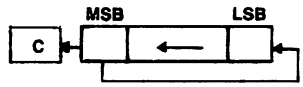


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

ROL=Rotate Left

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 0 0 0 r/m

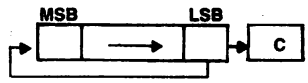


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

ROR=Rotate Right

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 0 0 1 r/m

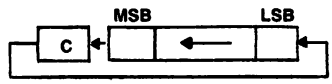


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

RCL=Rotate Through Carry Left

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 0 1 0 r/m

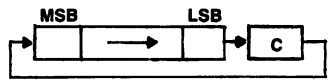


Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

RCR=Rotate Through Carry Right

Flags: O D I T S Z A P C
X X

1 1 0 1 0 0 v w	mod 0 1 1 r/m



Timing (clocks): single-bit register 2
single-bit memory 15+EA
variable-bit register 8+4/bit
variable-bit memory 20+EA+4/bit

AND=And

Flags: O D I T S Z A P C
O X X U X O

Reg./memory and register to either

0 0 1 0 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w=1

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 1 0 0 1 0 w	data	data if w=1

Timing: 4 clocks

Flags: O D I T S Z A P C
O X X U X O

TEST=And Function to Flags, No Result

Register/memory and register

1 0 0 0 0 1 0 w	mod reg r/m

Timing (clocks): register to register 3
register with memory 9+EA

Immediate data and register/memory

1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w=1

Timing (clocks): immediate with register 4
immediate with memory 10+EA

Immediate data and accumulator

1 0 1 0 1 0 0 w	data	data if w=1

Timing: 4 clocks

OR=Or

Flags: O D I T S Z A P C
O X X U X O

Reg./memory and register to either

0 0 0 0 1 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w=1

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 0 0 1 1 0 w	data	data if w=1

Timing: 4 clocks

XOR=Exclusive Or

Flags: O D I T S Z A P C
O X X U X O

Reg./memory and register to either

0 0 1 1 0 0 d w	mod reg r/m

Timing (clocks): register to register 3
memory to register 9+EA
register to memory 16+EA

Immediate to register/memory

1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w=1

Timing (clocks): immediate to register 4
immediate to memory 17+EA

Immediate to accumulator

0 0 1 1 0 1 0 w	data	data if w=1

Timing: 4 clocks

I.5.5 STRING MANIPULATION

REP=Repeat

Flags: O D I T S Z A P C

1 1 1 1 0 0 1 z

Timing: 6 clocks/loop

MOVS=Move String

Flags: O D I T S Z A P C

1 0 1 0 0 1 0 w

Timing: 17 clocks

CMPS=Compare String

Flags: O D I T S Z A P C
X X X X X X

1 0 1 0 0 1 1 w

Timing: 22 clocks

SCAS=Scan String

Flags: O D I T S Z A P C
X X X X X

1 0 1 0 1 1 1 w

Timing: 15 clocks

LODS=Load String

Flags: O D I T S Z A P C

1 0 1 0 1 1 0 w

Timing: 12 clocks

STOS=Store String

Flags: O D I T S Z A P C

1 0 1 0 1 0 1 w

Timing: 10 clocks

I.5.6 CONTROL TRANSFER

NOTE: Queue reintialization is not included in the timing information for transfer operations. To account for instruction loading, add 8 clocks to timing numbers.

CALL=Call

Flags: O D I T S Z A P C

Direct within segment

1 1 1 0 1 0 0 0	disp-low	disp-high

Timing: 11 clocks

Indirect within segment

1 1 1 1 1 1 1 1	mod 0 1 0 r/m

Timing: 13+EA clocks

Direct intersegment

1 0 0 1 1 0 1 0	offset-low	offset-high
	seg-low	seg-high

Timing: 20 clocks

Indirect intersegment

1 1 1 1 1 1 1 1	mod 0 1 1 r/m

Timing: 29+EA clocks

JMP=Unconditional Jump

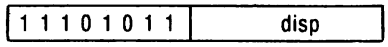
Flags: O D I T S Z A P C

Direct within segment

1 1 1 0 1 0 0 1	disp-low	disp-high

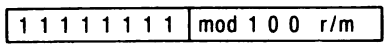
Timing: 7 clocks

Direct within segment-short



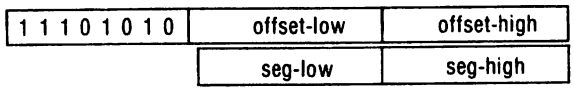
Timing: 7 clocks

Indirect within segment



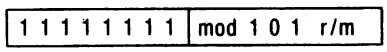
Timing: 7+EA clocks

Direct intersegment



Timing: 7 clocks

Indirect intersegment

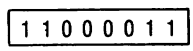


Timing: 16+EA clocks

RET=Return from CALL

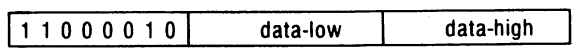
Flags: O D I T S Z A P C

Within segment



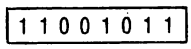
Timing: 8 clocks

Within seg. adding immediate to SP



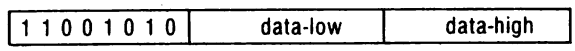
Timing: 12 clocks

Intersegment



Timing: 18 clocks

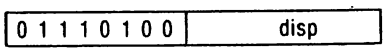
Intersegment, adding immediate to SP



Timing: 17 clocks

JE/JZ=Jump on Equal/Zero

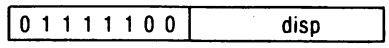
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JL/JNGE=Jump on Less/Not Greater or Equal

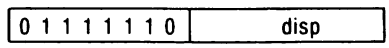
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JLE/JNG=Jump on Less or Equal/Not Greater

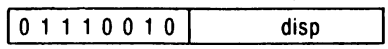
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JB/JNAE=Jump on Below/Not Above or Equal

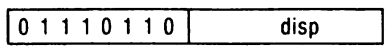
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JBE/JNA=Jump on Below or Equal/Not Above

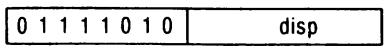
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JP/JPE=Jump on Parity/Parity Even

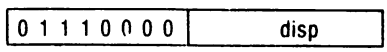
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JO=Jump on Overflow

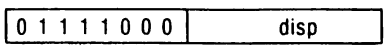
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JS=Jump on Sign

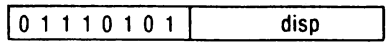
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNE/JNZ=Jump on Not Equal/Not Zero

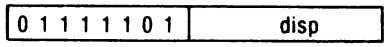
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNL/JGE=Jump on Not Less/Greater or Equal

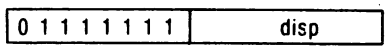
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNLE/JG=Jump on Not Less or Equal/Greater

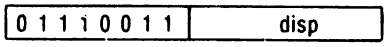
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNB/JAE=Jump on Not Below/Above or Equal

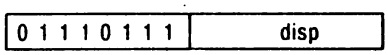
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNBE/JA=Jump on Not Below or Equal/Above

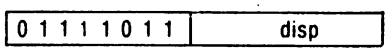
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNP/JPO=Jump on Not Parity/Parity Odd

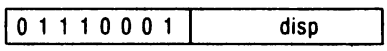
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

JNO=Jump on Not Overflow

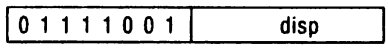
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not take 4

JNS=Jump on Not Sign

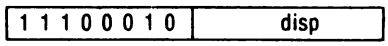
Flags: O D I T S Z A P C



Timing (clocks): jump is taken 8
jump is not taken 4

LOOP=Loop CX Times

Flags: O D I T S Z A P C



Timing (clocks): jump is taken 9
jump is not taken 5

LOOPZ/LOOPE=Loop While Zero/Equal

Flags: O D I T S Z A P C

1 1 1 0 0 0 0 1	disp

Timing (clocks): jump is taken 11
jump is not taken 5

LOOPNZ/LOOPNE=Loop While Not Zero/Not Equal

Flags: O D I T S Z A P C

1 1 1 0 0 0 0 0	disp

Timing (clocks): jump is taken 11
jump is not taken 5

JCXZ=Jump on CX Zero

Flags: O D I T S Z A P C

1 1 1 0 0 0 1 1	disp

Timing (clocks): jump is taken 9
jump is not taken 5

8086 Conditional Transfer Operations

INSTRUCTION	CONDITION	INTERPRETATION
JE or JZ	ZF=1	"equal" or "zero"
JL or JNGE	(SR xor OF)=1	"less" or "not greater or equal"
JLE or JNG	((SP xor OF) or ZF)=1	"less or equal" or "not greater"
JB or JNAE	CF=1	"below" or "not above or equal"
JBE or JNA	(CF or ZF)=1	"below or equal" or "not above"
JP or JPE	PF=1	"parity" or "parity even"
JO	OF=1	"overflow"
JS	SF=1	"sign"
JNE or JNZ	ZF=0	"not equal" or "not zero"
JNL or JGE	(SF xor OF)=0	"not less" or "greater or equal"
JNLE or JG	((SF xor OF) or ZF)=0	"not less or equal" or "greater"
JNB or JAE	CF=0	"not below" or "above or equal"
JNBE or JA	(CF or ZF)=0	"not below or equal" or "above"
JNP or JPO	PF=0	"not parity" or "parity odd"
JNO	OF=0	"not overflow"
JNS	OF=0	"not sign"

NOTE: "Above and below" refer to the relation between two unsigned values, while "greater" and "less" refer to the relation between two signed values.

INT=Interrupt

Flags: O D I T S Z A P C
O O

Type specified

1 1 0 0 1 1 0 1	type

Timing: 50 clocks

Type 3

1 1 0 0 1 1 0 0

Timing: 51 clocks

INTO=Interrupt on Overflow

Flags: O D I T S Z A P C
O O

1 1 0 0 1 1 1 0

Timing: 52 clocks if pass 4 clocks if fail

IRET=Interrupt Return

Flags: O D I T S Z A P C
R R R R R R R R R

1 1 0 0 1 1 1 1

Timing: 24 clocks

I.5.7 PROCESSOR CONTROL

CLC=Clear Carry

Flags: O D I T S Z A P C
O

1 1 1 1 1 0 0 0

Timing: 2 clocks

STC=Set Carry

Flags: O D I T S Z A P C
1

1 1 1 1 1 0 0 1

Timing: 2 clocks

CMC=Complement Carry

Flags: O D I T S Z A P C
X

1 1 1 1 0 1 0 1

Timing: 2 clocks

NOP=No Operation

Flags: O D I T S Z A P C

1 0 0 1 0 0 0 0

Timing: 3 clocks

CLD=Clear Direction

Flags: O D I T S Z A P C
O

1 1 1 1 1 1 0 0

Timing: 2 clocks

STD=Set Direction

Flags: O D I T S Z A P C
1

1 1 1 1 1 1 0 1

Timing: 2 clocks

CLI=Clear Interrupt

Flags: O D I T S Z A P C
0

1 1 1 1 1 0 1 0

Timing: 2 clocks

STI=Set Interrupt

Flags: O D I T S Z A P C
1

1 1 1 1 1 0 1 1

Timing: 2 clocks

HLT=Halt

Flags: O D I T S Z A P C

1 1 1 1 0 1 0 0

Timing: 2 clocks

WAIT=Wait

Flags: O D I T S Z A P C

1 0 0 1 1 0 1 1

Timing: 3 clocks

LOCK=Bus Lock Prefix

Flags: O D I T S Z A P C

1 1 1 1 0 0 0 0

Timing: 2 clocks

ESC=Escape (To External Device)

Flags: O D I T S Z A P C

1 1 0 1 1 x x x	mod x x x r/m

Timing: 7+EA clocks

NOTES:
If d=1 then "to"; if d=0 then "from."
If w=1 then word instruction; if w=0 then byte instruction.
If s:w=01 then 16 bits of immediate data form the operand.
If s:w=11 then an immediate data byte is sign extended to form the 16-bit operand.
If v=0 then "count"=1; if v=1 then "count" in (CL).
X=don't care.
Z is used for some string primitives to compare with ZF FLAG.
AL=8-bit accumulator.
AX=16-bit accumulator.
CX=Count register.
DS=Data segment.
DX=Variable port register.
ES=Extra segment.
Above/below refers to unsigned value.
Greater=more positive signed values.
Less=less positive (more negative) signed values.
See section I.2 for Operand summary.
See section I.4 for Segment Override summary.

I.6 PROCESSOR RESET REGISTER INITIALIZATION

Flags=0000H (to disable interrupts and single-stepping)

CS=FFFFH (to begin execution at FFFF0H)
IP=0000H

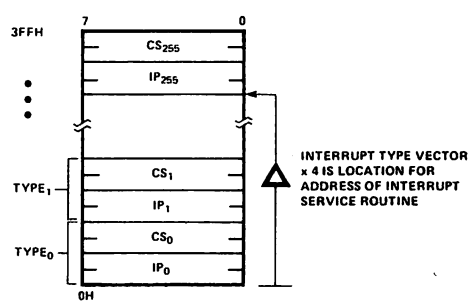
DS=0000H
SS=0000H
ES=0000H

No other registers are acted upon during reset.

I.7 8088 RESERVED LOCATIONS

INTERRUPT	LOCATION	FUNCTION
0	00H-03H	Divide by zero
1	04H-07H	Single step
2	08H-0BH	Non-maskable interrupt
3	0CH-0FH	One-byte interrupt instruction
4	10H-13H	Interrupt on overflow

Interrupt Pointer Table



I.8 8088 INSTRUCTION SET MATRIX

NOTES:

b=byte operation
d=direct
f=from CPU reg
i=immediate
ia=immed. to accum.
id=indirect
is=immed. byte, sign ext.
l=long ie. intersegment
m=memory
r/m=EA is second byte
si=short intrasegment
sr=segment register
t=to CPU reg
v=variable
w=word option
z=zero

I.9 MNEMONIC INDEX

```
PL/M-86 COMPILER     SIRIUS Systems Technology, Inc. (c) 1982  S-1 Hardware        04/01/82            PAGE   1
                     Example software drivers for S-1 Hardware

SERIES-III PL/M-86 V1.0 COMPILATION OF MODULE HARDWARE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  P.86 TEMP.SRC OPTIMIZE(3) PAGELENGTH(42) PAGEWIDTH(109) PRINT(:F4:HW.LS) NOOBJECT


                $TITLE                                  (c) 1982
                $SUBTITLE ('Example software drivers for S-1 Hardware')
                 /*****************************************************************************
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *                                                                           *
                  *****************************************************************************/


PL/M-86 COMPILER                                                                    04/01/82            PAGE   2
                     Example software drivers for S-1 Hardware

                $eject
                $SMALL ROM

   1            Hardware: do;

   2   1          Declare dcl       literally 'declare';
   3   1         Dcl     lit       literally 'literally';
   4   1          Dcl     addr      lit       'address',
                       ext       lit       'external',
                       init      lit       'initial',
                       intg      lit       'integer',
                       proc      lit       'procedure',
                       ptr       lit       'pointer',
                       pub       lit       'public',
                       rent      lit       'reentrant',
                       ret       lit       'return',
                       struc     lit       'structure',
                       boolean   lit       'byte',
                       true      lit       '0FFH',
                       false     lit       '0000H';


PL/M-86 COMPILER                                                                    04/01/82            PAGE   3
                     KB: Hardware bit defs

                $subtitle('KB: Hardware bit defs')


   5   1      dcl SR$intbit              lit '4';   /* KB shift register interrupt mask in6522 IER/IFR */
   6   1      dcl SR$enable              lit '0ch';          /* KB shift register enable in 6522 ACR */
   7   1      dcl CB1$intbit             lit '10h';       /* KB RDY edge-sense interrupt mask 6522 PCR */
   8   1      dcl CB1$pos_edge           lit '10h';           /* KB RDY edge-sense control in 6522 PCR */

   9   1      dcl kb$databit             lit '40h';            /* KB DATA level                        */
  10   1      dcl kb$ackctl              lit '2';              /* KB ACK control for 6522 output       */
  11   1      dcl kb$TIMEOUT             lit '300';            /* error timeout in milliseconds        */
  12   1      dcl timer1_ena             lit '0c0h';         /* timer 1 interrupt mask in 6522 IER/IFR */
```

```
$eject

/*  KYBRD PORT (e8040..e804f)  */

  dcl via(16)  struc(                          /* 6522 port organization              */

                          RB      byte,
                          RA      byte,
                          DDRB    byte,
                          DDRA    byte,
                          TIMER1  word,
                          TIMER1L word,
                          TIMER2  word,
                          SR      byte,
                          ACR     byte,
                          PCR     byte,
                          IFR     byte,
                          IER     byte,
                          RAX     byte) at(0e8000h);

  dcl kb$state              byte;              /* current state of keyboard stateware  */
  dcl kb$data               byte;              /* constructed data from keyboard       */

                                            /* nybble convert table for inverted shift reg */
  dcl Ctable(*) byte data (0,8,4,0ch, 2,0ah,6,0eh, 1,9,5,0dh, 3,0bh,7,0fh);

  dcl tick                lit '50';            /* console clock rate in milliseconds   */
```



```
$subtitle('KB: external routines')

  /*
   *    signal user about keyboard error state -- ring bell
   */
  dcl signal$KB$error lit 'Ringbell';
     /* Ringbell found in SOUND module */

  /*
   *    Process key board event -- in external module
   */
Process$Event: proc(event) byte ext;
  dcl event byte;
end;


  /*
   *   Software clock resource -- set timeout for interrupt to KB$reset
   */

set$KB$clock: proc(Period) ext;
  dcl Period intg;                             /* timeout delay in milliseconds        */
end set$KB$clock;
```



```
$subtitle('KB: Keyboard Stateware')
  /*
   *    KB interrupt entry  (level 6)
   */
kb$irq: proc pub rent;
   do case kb$state;
  /*
   *    state 0 to state 1:  shift register (full) interrupt
   */
  kbst0: do;
       via(4).ACR= via(4).ACR and not SR$enable; /* disable shift register          */
                                  /* prepare for interrupt on negative edge of KB RDY */
       via(4).PCR= via(4).PCR and not CB1$pos_edge;
       via(4).IER= 80h or CB1$intbit;
      disable;                                 /* time critical section             */
       kb$data = via(4).SR;                    /* get KB data from SR (clears SR IRQ) */
```

```
33   4              via(4).IER= SR$intbit;                    /* disable SR interrupt                */
                                                              /* assert KB ACK control on interrupt  */
34   4              via(4).RB = via(4).RB or kb$ackctl;       /* (CB1 IRQ is reset)                  */
35   4             enable;                                    /* end of critical section             */
36   4              kb$state = 1;                             /* set to state 1                      */
37   4            end;
                 /*
                  *    state 1 to state 2:  interrupt from negative edge on KB$RDY
                  */
38   3           kbst1: do;
39   4                disable;                                /* time critical section               */
40   4                if (via(4).RA and kb$databit) [| 0 then /* if data bit is not low then          */
41   4                    call kb$error;                      /* stop bit error has occurred         */
42   4                 else do;                    /* prepare for interrupt on positive edge of KB RDY */
43   5                    via(4).PCR= via(4).PCR or CB1$pos_edge;
                                                              /* release KB ACK control on interrupt */
44   5                    via(4).RB = via(4).RB and not kb$ackctl; /* (CB1 IRQ is reset)             */
45   5                    kb$state = 2;                       /* set to state 2                      */
46   5                  end;
47   4                enable;                                 /* end of critical section             */
48   4               end;
```



```
               $eject


                 /*
                  * state 2 to state 0:  interrupt from positive edge on KB$RDY
                  */
49   3           kbst2: do;
50   4                if (via(4).RA and kb$databit) = 0 then  /* if data bit is low then             */
51   4                    call kb$error;                      /* stop bit error has occurred         */
52   4                 else do;
53   5                    call kb$reset;                     /*reset hardware/software for next event */
                                        /* call event processing routine with order of bits reversed to */
                                        /*   reflect physical key number and event type (open or close) */
54   5                    if not Process$Event( shl(Ctable(kb$data and 0fh),4)
                              or Ctable(shr(kb$data,4)) ) then
55   5                         call signal$KB$error;          /* signal error in event process       */
56   5                  end;
57   4               end;

58   3           end;

59   2         end kb$irq;
```



```
               $subtitle('KB: Keyboard support routines')

60   1         kb$reset:  proc rent;                       /* puts KB hardware/software into state 0 */
61   2           dcl dummy byte;

62   2              via(4).IER = CB1$intbit;                 /* clear CB1 interrupts                */
63   2              via(4).RB  = via(4).RB and not kb$ackctl; /* release kb$ack                      */
64   2              via(4).ACR = via(4).ACR or SR$enable;    /* enable shft reg                     */
65   2              dummy      = via(4).SR;                  /* clr any pending irq                 */
66   2              via(4).IER = 80h or SR$intbit;           /* enable sr interrupts                */
67   2              kb$state   = 0;                          /* init keybrd state                   */
68   2              call set$KB$clock(0);                    /* clear timeout counter               */
69   2         end kb$reset;

70   1         kb$error:  proc rent;
71   2              via(4).RB  = via(4).RB or kb$ackctl;     /* force kb$ack high                   */
72   2              via(4).IER = 7fh;                        /* allow no interrupts                 */
73   2              call set$KB$clock(kb$TIMEOUT);           /* time out keyboard                   */
74   2         end kb$error;

75   1         kb$init:  proc  pub rent;

76   2              via(4).RB   = via(4).RB and (0FFh-3);
77   2              via(4).DDRA = via(4).DDRA and not kb$databit;
78   2              via(4).DDRB = via(4).DDRB or kb$ackctl;
79   2              via(4).IER  = 7fh;
80   2              via(4).PCR  = 0;
81   2              via(4).ACR  = 0;
```

```
 82   2          via(2).ACR= (via(2).ACR and 0c0h) or 40h;
 83   2          via(2).timer1L= tick*1000;
 84   2          via(2).IER  = timer1_ena and 7fh;
 85   2          call kb$reset;
 86   2        end kb$init;
```



```
                $SUBTITLE ('CRTreg: controller chip registers')

 87   1      DCL   CRT$0     byte   AT (0E8000H);              /* CRT-chip address register          */
 88   1      DCL   CRT$1     BYTE   AT (0E8001H);              /* CRT-chip internal register port    */


                /*
                 *      Set CRT register
                 */
 89   1      set$CRT$reg: proc (reg,value) rent;
 90   2        dcl reg byte;
 91   2        dcl value byte;
 92   2          CRT$0= reg;                                   /* select register                    */
 93   2          CRT$1= value;                                 /* set data                           */
 94   2        end set$CRT$reg;
```



```
                $SUBTITLE ('CRTreg: cursor-display mode control')

 95   1      dcl   rast$start  lit    '10';            /* CRT reg: cursor-start & cursor-display mode */

 96   1       DCL   Cursor$PAR   BYTE; /* VAR: contents for CRT cursor-start raster & cursor display mode */
 97   1       dcl   blink$on      boolean;                  /* FLAG:  =0  Blinking cursor on (fast) */
 98   1       dcl   curs$off      boolean;                  /* FLAG: [|0  Cursor off                */

                /*
                 *      Set cursor to current Cursor parameter byte.
                 */
 99   1      set$cursor: proc rent;
100   2          call set$CRT$reg(rast$start,Cursor$PAR);      /* set raster start reg               */
101   2        end set$cursor;

                /*
                 *      Set block cursor.
                 */
102   1      BLOCK$CRS:PROC  RENT;
103   2          Cursor$PAR = Cursor$PAR AND 0E0h;             /* set block cursor                   */
104   2          call set$cursor;                              /* set cursor mode reg                */
105   2        END BLOCK$CRS;

                /*
                 *      Set underscore cursor.
                 */
106   1      UNDERSCORE$CRS:PROC RENT;
107   2          Cursor$PAR = 00Fh OR (Cursor$PAR AND 0E0h);   /* set underscore cursor              */
108   2          call set$cursor;                              /* set cursor mode reg                */
109   2        END UNDERSCORE$CRS;
```



```
                $eject
                /*
                 *    Return cursor to previous modes: block or underline, steady or flashing
                 */
110   1      CURSOR$ON:PROC  RENT;
111   2          curs$off= false;                              /* reset cursor off flag              */
112   2          if blink$on then Cursor$par= Cursor$par or 060h; /* set to flashing mode            */
114   2           else Cursor$par= Cursor$par and 01Fh;        /* set to steady mode                 */
115   2          call set$cursor;                              /* set cursor mode reg                */
116   2        END CURSOR$ON;
```

```
                /*
                 *      Turn cursor off.
                 */
 117    1       CURSOR$OFF:PROC  RENT;
 118    2           curs$off= true;                                 /* set cursor off flag                */
 119    2           Cursor$PAR = 020h OR (Cursor$PAR AND 01Fh);     /* set to off mode                    */
 120    2           call set$cursor;                                /* set cursor mode reg                */
 121    2         END CURSOR$OFF;

                /*
                 *      Set cursor blinking.
                 */
 122    1       CRS$BLINK$ON:PROC  RENT;
 123    2           blink$on= true;                                 /* set blinking on flag               */
 124    2           if not curs$off then Cursor$PAR= 060h OR Cursor$PAR; /* set flashing,if not off       */
 126    2           call set$cursor;                                /* set cursor mode reg                */
 127    2         END CRS$BLINK$ON;
                /*
                 *      Set cursor steady.
                 */
 128    1       CRS$BLINK$OFF:PROC  RENT;
 129    2           blink$on= false;                                /* reset blinking on flag             */
 130    2           if not curs$off then Cursor$PAR= 01Fh and Cursor$PAR; /* set steady,if not off         */
 132    2           call set$cursor;                                /* set cursor mode reg                */
 133    2         END CRS$BLINK$OFF;
```


CRTreg: Cursor positioning

```
                $SUBTITLE ('CRTreg: Cursor positioning')

 134    1       dcl   cursaddrH   lit    '14'; /* CRT reg: MSByte of cursor location word, bits: xx54$3210 */
 135    1       dcl   cursaddrL   lit    '15';                 /* CRT reg: LSByte of cursor location word */


                  /*
                   *   Position Cursor to Absolute Font Cell number
                   *     and display bank
                   */

 136    1       POS$Cursor: proc (Cell$number) pub rent;
 137    2         dcl Cell$Number word;                         /* Absolute Font Cell Number & diplay bank */
 138    2           call set$CRT$reg (cursaddrL, low(Cell$number));
 139    2           call set$CRT$reg (cursaddrH, high(Cell$number));
 140    2         end POS$Cursor;
```


CRT: video contrast & brightness

```
                $SUBTITLE ('CRT: video contrast & brightness')

 141    1       DCL   CBctrl    BYTE   AT (0E8040H);           /* Contrast & Brightness control register */
                                                                /* bits: CCCB$BB--                    */

                /*
                 *      Raise video contrast one level.
                 */
 142    1       contrast$up: proc rent;
 143    2         dcl a byte;
 144    2         if (a:= (CBctrl + 20h) and 0E0h) [| 0 then    /* add & check upper limit            */
 145    2           CBctrl= (CBctrl and 01FH) or a;             /* set contrast, bits: 765            */
 146    2       end contrast$up;
                /*
                 *      Lower video contrast one level.
                 */
 147    1       contrast$down: proc rent;
 148    2         dcl a byte;
 149    2         if (a:= (CBctrl - 20h) and 0E0h) [| 0E0h then /* sub & check lower limit            */
 150    2           CBctrl= (CBctrl and 01FH) or a;             /* set contrast, bits: 765            */
 151    2       end contrast$down;

                /*
                 *      Raise video brightness one level.
                 */
 152    1       bright$up: proc rent;
 153    2         dcl a byte;
 154    2         if (a:= (CBctrl + 4) and 01CH) [| 0 then      /* add & check upper limit            */
 155    2           CBctrl= (CBctrl and 0E3H) or a;             /* set brightness, bits: 432          */
 156    2       end bright$up;
```

```
                /*
                 *      Lower video brightness one level.
                 */
157   1       bright$down: proc rent;
158   2         dcl a byte;
159   2         if (a:= (CBctrl - 4) and 01Ch) [| 01Ch then     /* sub & check lower limit          */
160   2           CBctrl= (CBctrl and 0E3H) or a;               /* set brightness, bits: 432        */
161   2       end bright$down;
```


CRT: display RAM/Font Cells

```
                $SUBTITLE ('CRT: display RAM/Font Cells')

162   1       dcl   screen$ram word at (0F0000h);             /* memory address of display RAM    */
163   1       dcl   screen$addr  ptr;                       /* display ram pointer, base of word ARRAY */
164   1       DCL   SCREEN based screen$addr (2000) word;     /* ARRAY of Font Cell Pointers      */

                  /*
                   *    Screen Buffer Word variables
                   */

165   1       dcl   char$mode     word     pub;               /* CRT attribute bits: 7654$3---    */

166   1       dcl   char$base     word     pub;               /* CRT Font Cell Pointer base for   */
                                                            /*      ASCII symbol index          */

167   1        DCL   REVBIT      LIT    '8000H';
168   1        DCL   BGBIT       LIT    '4000H';
169   1        DCL   UNDBIT      LIT    '2000H';
170   1        dcl   INVBIT      lit    '1000h';
171   1        dcl   extraBIT    lit    '0800h';


                 /*
                  *  Display symbol from character set (typically ASCII)
                  *     at absolute Font Cell number
                  *      (typically: [line| * [display width| + [column| )
                  *     with current Cursor & Display modes.
                  */

172   1       Display$symbol: proc (Symbol$code,Cell$number) pub rent;
173   2         dcl Symbol$code byte;                         /* Symbol print code                */
174   2         dcl Cell$Number word;                         /* Absolute Font Cell Number        */
175   2           screen(Cell$Number)= (Symbol$code + char$base) OR char$mode;
176   2         end Display$symbol;
```


CRT hardware initialization

```
                $SUBTITLE ('CRT hardware initialization')
                                                            /* COMMENT THIS !!!!                */
177   1       DCL CRT$config (*) BYTE DATA (92,80, 81,0CFh, 25,6, 25,25, 3,14, 0,15, 0,0, 0,0);


178   1       CRT$Init:  PROC;
179   2         DCL  I  BYTE;

180   2         screen$addr= @screen$ram;

181   2         char$mode= BGBIT;
182   2         char$base= 20;

183   2         curs$off= false;
184   2         blink$on= false;
185   2         Cursor$PAR= 0;

186   2         DO I=0 TO 0FH;
187   3           CALL SET$CRT$REG (I,(CRTconfig(I)));
188   3           END;

189   2         END CRT$Init;
```

```
$SUBTITLE ('SOUND variables & hardware defs')


dcl bell$freq LIT    '76';                    /* period of bell tone: frequency= 14.9KHz */

dcl     codec$clk       word at (0E8084h);      /* TIMER1: codec clock frequency        */
dcl     codec$ctl       byte at (0E808Bh);      /* ACR: codec clock control register    */
dcl     codec$sda       word at (0E8060h);      /*                                      */
dcl     volume          byte at (0E802Ah);      /* SR: volume shift-register            */
dcl     vol$ctl         byte at (0E802Bh);      /* ACR: SR control register             */
dcl     vol$clk         word at (0E8028h);      /* TIMER2: volume SR clock              */

dcl bell$on   byte;                             /* FLAG: bell sound presently active    */
dcl vol$level byte;    /*current volume level (nine levels: 0 --| 8)
                                                  /* volume shift pattern lookup table    */
dcl vol$table (*) byte data (0FFh,7FH,3FH,1FH,0FH,7,3,1,0);
```



```
$SUBTITLE ('SOUND: Bell control')

  /*
   *   Software clock resource -- set timeout for interrupt to Bell$clock
   */

set$BELL$clock: proc (Period) ext;
  dcl Period intg;                              /* timeout delay in milliseconds        */
  end set$BELL$clock;


  /*
   *    CODEC Hardware reset
   */

Bell$init: proc pub rent;
   vol$level= length(vol$table)-2;              /* set initial volume level near max    */
   call Bell$clock;                            /* set hardware to a known & quiet state */
  end Bell$init;
```



```
$eject

Bell$clock:     proc  pub rent;

   codec$ctl = codec$ctl and not 0C0h;          /* disable codec clock                  */

   codec$sda = 5E00h;                           /* initialize codec SDA to input mode... */
   codec$sda = 0D40h;                           /* ... to reduce extraneous noise       */
   codec$sda = 0AA80h;
   codec$sda = 00C0h;

   vol$ctl = (vol$ctl and not 3Ch) or 10h;      /* set SR & T2 volume register modes    */
   vol$clk = 1;                          /* volume clock frequency set beyond perception */
   volume = vol$table(vol$level);               /* set volume to current level          */
   bell$on = false;                             /* set bell state to off                */

   end bell$clock;

Ring$bell: proc pub rent;
    if not bell$on then do;                     /* start bell if sound is off           */
     call bell$clock;                             /* init codec hardware on every bell    */
     codec$sda = 0f80h;                         /* set output waveform to 4 up & 4 down, */
                                                /*    a low amplitude triangle wave.    */
     codec$ctl = codec$ctl or 0c0h;               /* set codec clock to free run          */
     codec$clk = bell$freq;                       /* set audio pitch frequency            */
     bell$on = true;                              /* set bell state on                    */
       end;
   call set$bell$clock(100);                    /* turn off bell in 100 milliseconds    */
 end;
```

```
                $SUBTITLE ('SQUND: volume control')
                /*
                 *      Raise CODEC volume one level.
                 */
 229    1       volume$up: proc rent;
 230    2           if vol$level |= length(vol$table)-1 then      /* check upper limit                */
 231    2               vol$level= length(vol$table)-1;           /* set to max volume                */
 232    2            else vol$level= vol$level+1;                 /* bump level up by one             */
 233    2               volume= vol$table(vol$level);             /* set volume register              */
 234    2        end volume$up;

                /*
                 *      Lower CODEC volume one level.
                 */
 235    1       volume$down: proc rent;
 236    2           if vol$level |= length(vol$table)-1 then      /* check upper limit                */
 237    2               vol$level= length(vol$table)-2;           /* set to max volume-1              */
                    else
 238    2               if vol$level[|0 then vol$level= vol$level-1; /* drop level by one             */
                    volume= vol$table(vol$level);                 /* set volume register           */
 241    2       end volume$down;
```



```
                $subtitle('SIO: Serial I/O dvrs for TTY: and UL1:')

                /*ctr device dcls*/
 242    1       dcl sioctr struc
                    (adata byte,
                     bdata byte,
                     xxx   byte,
                     ctrctl byte) at (0E0020h);
                /*sio device dcls*/
 243    1       dcl siodev struc
                    (adata byte,
                     bdata byte,
                     actl  byte,
                     bctl  byte) at(0E0040h);
 244    1       dcl rx$avail literally '1',
                    tx$empty literally '4';

 245    1       dcl serial_params struc
                    (actrlsb byte,                                /*LSByte of chan a.'s baud rate    */
                     actrmsb byte,                                /*MSByte ...                       */
                     bctrlsb byte,                                /*LSByte of chan b.'s baud rate    */
                     bctrmsb byte,                                /*MSByte ...                       */
                       /* if [baud| then lsb = ??h  msb = ??h 1.25Mhz/([baud|*16)
                              50 ===|        1Ah        06h     50.00  -0-    (min.tol.dist.43.75%)
                              75 ===|        11h        04h     75.00  -0-    (     "       43.75%)
                             110 ===|        C6h        02h    110.00  -0-    (     "       43.75%)
                             134.5 ===|      44h        02h    134.00  -0.37% (     "       40.23%)
                             150 ===|        08h        02h    150.00  -0-    (     "       43.75%)
                             200 ===|        86h        01h    200.00  -0-    (     "       43.75%)

                             300 ===|        04h        01h    300.00  -0-    (     "       43.75%)

                             600 ===|        82h        00h    600.00  -0-    (     "       43.75%)
                            1.2k ===|        41h        00h   1201.00  +0.08% (     "       42.99%)
```



```
                $eject

                ------------------------------------------------
                                        2Ch        00h   1775.00  -1.39% (     "       30.54%)
                            1.8k ===|   2Bh        00h   1816.00  +0.09% (     "       42.88%)
                ------------------------------------------------
                                        28h        00h   1953.00  -2.36% (     "       21.33)
                            2.0k ===|   27h        00h   2003.00  +0.15% (     "       42.32)
                ------------------------------------------------
                            2.4k ===|   21h        00h   2367.00  -1.38% (     "       30.64%)
                                        20h        00h   2441.00  +1.71% (     "       27.51%)
                ------------------------------------------------
```

```
                3.6k ===|       16h       00h   3551.00  -1.36% (      "       30.83%)
                                15h       00h   3720.00  +3.33% (      "       12.4%)
-------------------------------------------------
                                11h       00h   4595.00  -4.27% (      "       3.185%)
                4.8k ===|       10h       00h   4882.00  +1.02% (      "       34.06%)
-------------------------------------------------
                                09h       00h   8680.55  -9.58% (DISTORTED)
                9.6k ===|       08h       00h   9765.56  +1.73% (min.tol.dist.27.32%)
-------------------------------------------------
                                06h       00h  13020.83  -9.58% (DISTORTED)
                                05h       00h  15625.00  +8.51% (DISTORTED)
-------------------------------------------------
                                05h       00h  15625.00 -18.62% of 19.2k (DISTORTED)
              19.2k ===|        04h       00h  19531.25  +1.02% (min.tol.dist.34.06%)

min.tol.dist. figure assumes no channel noise effects.
  NOTE: possible noise DOES NOT includes bias distorition
        caused by various cable capacitance effects*/
```



```
$eject

     cr2a    byte,      /*bus interface option: 10h if baud a [= baud b
                                                 14h if baud a | baud b*/

     cr4a    byte,
     cr4b    byte,
       /*cr4x (16x)$54$(stops)$(even)$(parenb) = 4?h
              01    00 ss      e      p
                       ss = 01 1 stop
                          = 10 1.5 stop
                          = 11 2 stop
                               e = 1 even
                               e = 0 odd, byte transparent
                                     p = 1 even or odd
                                     p = 0 byte transparent*/

     cr3a    byte,
     cr3b    byte,
       /*cr3x (rbits)$(autoenb)$4$3$2$1$(renb) = ?1h
              bb       1         0 0 0 0  1
              bb = 11 byte transparent cr3x = E1h
                 = 01 even,odd         cr3x = 61h*/

     cr5a    byte,
     cr5b    byte) EXT;
       /*cr5x (dtr)$(tbits)$(br)$(tenb)$2$(rts)$0 = ?Ah
              1     bb      0    1     0  1    0
                    bb = 11 space,mark cr5x = EAh
                    bb = 01 even,odd,no cr5x = AAh*/
```



```
$subtitle('SIO: Serial I/O dvrs for port A -- TTY$INSTAT & TTY$STAT')

246   1     TTY$in$stat:proc boolean PUB;

247   2     if ( (siodev.actl AND rx$avail) [| 0)
            then return(true);
249   2     return(false);

250   2     end TTY$in$stat;


251   1     TTY$stat:proc boolean PUB;

252   2     if ( (siodev.actl AND tx$empty) = 0)
            then return(true);
254   2     return(false);

255   2     end TTY$stat;
```

```
$subtitle('SIO: Serial I/O dvrs for port A -- TTY$GET & TTY$PUT')

256   1     TTY$get:proc byte PUB;

            /*user must not activate this procedure if siodev chan. a reg. ptr
              is not set to 0 (only [| 0 if user has been mucking with hardware*/
257   2     do while( (siodev.actl AND rx$avail) = 0);      /*wait forever till empty      */
258   3     end;
259   2     return(siodev.adata);                           /*input form 7201              */

260   2     end TTY$get;


261   1     TTY$put:proc(char) PUB;
262   2     dcl char byte;

            /*user must not activate this procedure if siodev chan. a reg. ptr
              is not set to 0 (only [| 0 if user has been mucking with hardware*/
263   2     do while( (siodev.actl AND tx$empty) = 0);      /*wait forever till empty      */
264   3     end;
265   2     siodev.adata = char;                            /*output a char                */
266   2     return;

267   2     end TTY$put;
```



```
$subtitle('SIO: Serial I/O dvrs for port B -- UL1$STAT & UL1PUT')

268   1     UL1$stat:proc boolean PUB;

269   2     if ( (siodev.bctl AND tx$empty) = 0)
            then return(true);
271   2     return(false);

272   2     end UL1$stat;


273   1     UL1$put:proc(char) PUB;
274   2     dcl char byte;

            /*user must not activate this procedure if siodev chan. b reg. ptr
              is not set to 0 (only [| 0 if user has been mucking with hardware*/
275   2     do while( (siodev.bctl AND tx$empty) = 0);      /*wait forever till empty      */
276   3     end;
277   2     siodev.bdata = char;                            /*output a char                */
278   2     return;

279   2     end UL1$put;
```



```
            $subtitle('SIO: Serial I/O dvrs for ports A & B -- SIO$INIT')
280   1     SIO$init:proc PUB;
281   2     siodev.actl = 00$011$000b;                      /*chan. a reset                */
282   2     siodev.bctl = 00$011$000b;                      /*chan. b reset                */

            /*load timer now; cant touch 7201 chip for 4 2.5Mhz clocks*/
283   2     sioctr.ctrctl = 36h;                            /*7$(ctra)$(rl)$(mode)$(bin)   */
284   2     sioctr.adata = serial_params.actrlsb;
285   2     sioctr.adata = serial_params.actrmsb;
286   2     sioctr.ctrctl = 76h;                            /*7$(ctrb)$(rl)$(mode)$(bin)   */
287   2     sioctr.bdata = serial_params.bctrlsb;
288   2     sioctr.bdata = serial_params.bctrmsb;

            /*cr2a bus interface option*/
289   2     siodev.actl = 2;                                /*--|cr4a                      */
290   2     siodev.actl = serial_params.cr2a;

            /*cr4x*/
```

```
291   2       siodev.actl = 4;                                /*--|cr4a                */
292   2       siodev.actl = serial_params.cr4a;
293   2       siodev.bctl = 4;                                /*--|cr4b                */
294   2       siodev.bctl = serial_params.cr4b;

              /*cr3x*/
295   2       siodev.actl = 3;                                /*--|cr3a                */
296   2       siodev.actl = serial_params.cr3a;
297   2       siodev.bctl = 3;                                /*--|cr3b                */
298   2       siodev.bctl = serial_params.cr3b;
```



```
              $eject

              /*cr5x*/
299   2       siodev.actl = 5;                                /*--|cr5a                */
300   2       siodev.actl = serial_params.cr5a;
301   2       siodev.bctl = 5;                                /*--|cr5b                */
302   2       siodev.bctl = serial_params.cr5b;

              /*cr0x reset ext/st intrs to enable modem control sense--| autoenb chans.
                also --| crlx, set intr params*/
303   2       siodev.actl = 00$010$001b;
304   2       siodev.actl = 0;                                /*no intrs              */
305   2       siodev.bctl = 00$010$001b;
306   2       siodev.bctl = 0;                                /*no intrs              */

307   2       end sio$init;
```



```
              $subtitle ('PPORT -- centronics interface routines ')

              /*
               * This module implements the initialization, LISTST, and LIST functions
               * for a Centronics-compatible parallel printer interface, using the
               * 6522 VIA chip.
               *
               * Our entry points are named pp$init, LPT$stat, and LPT$put respectively,
               * it's up to our caller to decode the I/O byte and call the appro-
               * riate routines.
               */
```



```
              $eject
308   1       declare pp$base pointer;                        /* baseaddr for a 6522   */
309   1       declare pp based pp$base structure (            /* 6522 template         */
                      rb byte,                                /* out-in reg 'b'        */
                      ra byte,                                /* out-in reg 'a'        */
                      ddrb byte,                              /* data-direction, reg 'b'*/
                      ddra byte,                              /* data-direction, reg 'a'*/
                      tlcl byte,                              /* tl ctr(r)/lat(w) lo   */
                      tlch byte,                              /* tl ctr hi             */
                      tlll byte,                              /* tl latch lo           */
                      tllh byte,                              /* tl latch hi           */
                      t2cl byte,                              /* t2 ctr(r)/lat(w) lo   */
                      t2ch byte,                              /* t2 ctr hi             */
                      sr byte,                                /* shift register        */
                      acr byte,                               /* auxiliary ctrl reg    */
                      pcr byte,                               /* peripheral ctrl reg   */
                      ifr byte,                               /* interrupt flg register*/
                      ier byte,                               /* interrupt enbl register*/
                      rax byte                                /* out-in reg 'a' NO HANDSHAKE */
              );
              /*
               * Bit definitions for Centronics-style parallel interface, 'vial'.
               */
310   1       declare vial$base literally '0e8020h';          /* baseaddr for this chip*/
311   1       declare ds$l literally '01h';                   /* data strobe (pb0)     */
312   1       declare pi$h literally '02h';                   /* this datum for vfu (pbl)*/
313   1       declare bz$h literally '20h';                   /* printer busy (pb5)    */
```

```
314   1      declare ak$l literally '40h';                  /* printer ack (pb6)                */
315   1      declare sl$h literally '80h';                  /* on-line and no error (pb7)       */
             /*
              * Bit definitions for multi-use pio, 'via2'.
              */
316   1      declare via2$base literally '0e8040h';         /* baseaddr for this chip           */
317   1      declare te$h literally '01h';                  /* talk-enable line                 */
```



```
                PPORT -- centronics interface routines

             $eject
             /*
              * initial setup for parallel printer port
              * Note we use via2 during this setup to get talk-enable turned on, and
              * thus someone MUST ALREADY HAVE VIA2 INITIALIZED.
              */

318   1      pp$init: procedure public;
319   2          pp$base = via2$base;                       /* point to secondary chip for te   */
320   2          pp.rb = pp.rb or te$h;                     /* set 'talk enbl'                  */
321   2          pp$base = via1$base;                       /* point struc at primary chip      */
322   2          pp.ra = 0;                                 /* ra is dataport, init with 0's    */
323   2          pp.ddra = 0ffh;                            /* set all ra bits as outgoing      */
324   2          pp.rb = ds$1;                              /* rb is ctrlport, init no ds/pi    */
325   2          pp.ddrb = ds$1 or pi$h;                    /* these 2 only are outgoing        */
                                                           /* cal/ca2 cb1/cb2 not used         */
                                                           /* timers/shiftreg not used         */
326   2      end pp$init;
```



```
             $eject

             /*
              * Test status of printer, return true if on-line and not busy, else
              * false.  For some reason, the Altos code explicitly deasserted data
              * strobe before testing; we'll assume that this represents an Altos
              * fubar and is not required here.
              */
327   1      LPT$stat: procedure byte public;
328   2          if (pp.rb and (sl$h or bz$h)) = sl$h then return 0ffh;
330   2          return 0;
331   2      end LPT$stat;


             /*
              * Put one character to the printer interface.
              */
332   1      LPT$put: procedure(ch) public;
333   2         declare ch byte;
334   2          do while LPT$stat = 0; end;                /* wait for printer ready           */
336   2          pp.ra = ch;                                /* put outgoing char on the port    */
337   2       disable;
338   2          pp.rb = pp.rb and not ds$1;                /* assert data strobe               */
339   2          pp.rb = pp.rb or ds$1;                     /* deassert data strobe             */
340   2       enable;
341   2          return;
342   2      end LPT$put;
```

```
            $SUBTITLE ('Example software drivers for S-1 Hardware')

 343   1    end Hardware;



MODULE INFORMATION:

     CODE AREA SIZE     = 073EH   1854D
     CONSTANT AREA SIZE = 0000H      0D
     VARIABLE AREA SIZE = 0014H     20D
     MAXIMUM STACK SIZE = 000EH     14D
     807 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

END OF PL/M-86 COMPILATION
```

INDEX

SUPPLEMENTAL TECHNICAL REFERENCE MATERIAL

APPLICATION NOTE: ØØ2

Revision Ø

First VICTOR printing March 1983.

CONTENTS

CONTENTS
continued